Paper Review - CSR Core Suprise Removal in Commodity Operating Systems

Nico Westerbeck

December 11, 2016

Abstract

Here you provide a very short intoduction into your topic and sum up your hand-in. It is important to highlight the main issues to be discussed in your hand-in here.

Contents

1 Introduction of the Research Field

The reviewed paper is of the field software fault tolerance, a field of computer science trying to allow execution in the presence of hard- and software faults and failures. A system is regarded failing if it cannot provide its service any more, which naturally is the result of a present flaw in the design making the system unable to cope with its situation. Less abstract, we want a computer to still work if something went wrong.

Researching related work in order to understand the novelties of your own ideas, presented in your upcoming Master Thesis, is a challenging task. As a preparation, your task is to research a topic on your own, which is given you by the paper. The task is not to repeat the original paper but to study its overriding principles and the research field. The hand-in should summarize the results of your research. In the introduction section, you show what the field is about from a high-level point of view. The introduction also motivates the topic.

2 Basics

Introduce general knowledge necessary to understand your paper. Here you should introduce concepts, ideas, definitions, terms, etc., which are not explicitly part of your topic, but are needed to understand it. This part is intended for readers which are not familiar with your topic at all [1], so keep it straight and simple.

3 Previous and Related Work

Currently there are no other topics addressing the handling of core failures, so there is no related work the focus paper can be compared to. However in this paper I will use several other papers to evaluate the content and give some alternative approaches. I researched work on the statistics behind hardware failures and

Study the history of your topic by investigating its related work. Compare the given paper to other publications dealing with the same research field. Describe the shortcoming of existing approaches that will be solved or improved by the paper of your topic.

This is also the place to cite all sources and papers you have used in this handin and in your presentation. BibTEX entries describing references are to be added to the `refs.bib` file. You can find pre-formatted BibTEX records in the Internet – e.g. using the ACM Digital Library or Google Scholar.

> "Sometimes you might want to use the *quotation* environment in order to cite larger passages of the related work."

4 The Topic's Approach

To understand the mechanism, it is essential to understand the failure model the paper assumes. They cope with spontaneous, permanent core failures that could result by malfunctioning transistors or violations of cpu-internal constraints. These failures might happen in a cascade, crashing several cores in a row within a short timespan. During a crash, the internal registers and buffers might be destroyed, however the paper requires the internal cache to be flushed upon failure.

The mechanism described in [SHP+16] is loosely based on the kernel hotplug mechanism for adding and removing cores during runtime. However, the hotplug mechanism in current kernels depends on the cooperation of the core that is to be shut down, which is obviously not possible in the case of a hardware failure. At first, the remaining system needs to notice the crash of one of the cores. For this, the paper requires a failure detection unit (FDU), which can either be implemented in hardware or software and reliably detects failures of one or more cores in the system. This FDU will trigger a 4-step migration procedure, migrating all data from the faulty core to another one

5 Evaluation

Present and discuss measurements, experiments, examples, ... but do not repeat the entire evaluation of the original paper. *Cite* all figures and tables copied form other papers. You can keep this section short and focus on the aspects that were improved by the paper compared to existing approaches.

6 Discussion

This paper offers the first approach for an operating system to survive in the presence of hardware failures. The introduced overhead is very low and in general including this mechanism into the

[1]However, you can use footnotes to introduce some more advanced concepts.

linux kernel brings very little disadvantages but some advantages. Still there are some points which lower the overall usefulness of the paper or can be critizised. Also there are some weaknesses which are already adressed within the paper

- A lot of kernel structures are assuming code to be executed (locks, interrupts)
- All load is migrated to core 0, which results in a temporary high-load situation of that core

I will not discuss the weaknesses that are already ruled out in the paper itself, as they I regard the discussions of these points as very complete. However, I have my own doubts about the usefulness and applicability of the approach, which I will cover in the next subsections.

6.1 HTM required to rollback

The approach has the requirement, that upon a processor fail within a HTM transaction, the HTM implementation will rollback the transaction. Combined with my concerns expressed in section 6.6, I doubt every implementation is actually able to provide that promise. I did however not find any literature to disprove or prove them.

6.2 FDU

One fundamental requirement of the work is a working failure detection unit, that can reliably detect if a core has failed and that will not stop working itself. However, in the paper there was very little discussion about the properties of this unit, except for saying that it can be implemented in hardware. This is indeed true, as there are several patents for failure detection units [CHLVP14] [Ohi09], however it is questionable if this unit can be assumed present on all machines. If not, this FDU would need to be emulated in software, bringing multiple drawbacks. At first, a software FDU would be bound to a process which is bound to a CPU which is assumed to be unreliable in this paper. So a solution with failover instances would need to be set up, which was not discussed in this paper. Also the FDU offers a potential vulnerability, enabling an attacker to fully disable the machine if taken over.

It is also imaginable of the FDU noticing false positives, shutting down an intact processor including the currently running process while that core could still be working or just being offline for a bound amount of time and taking over old execution again. If that should be the case, the recovery process would have assigned work to another core resulting in code being executed twice and race-conditions between the cores, likely resulting in data corruption or a crash.

6.3 Failures in Kernelspace

The paper states that it is not possible to reach 100% failure tolerance, as IPIs could get lost and lock structures inside the kernel leave a small chance of failures even when wrapped inside transactions. Both do not pose a statistically significant problem, as 99% of the locks committed successfully with some optimizations and the team was able to recover a big portion of IPIs, but there were some that were not recoverable, leaving a small chance of the kernel to fail even with the CSR measures in place. This is already partially discussed inside the paper but was not subject to an extensive debate, so I wanted to mention it again. But as 100% coverage is an unreachable goal in the area of software fault-tolerance, I don't think this poses a great problem.

The approach by the paper requires HTM to be present to tolerate kernel failures. However, HTM is a very new and untested feature. It is by far not present in all modern CPUs and thus greatly reduces the audience which can use this feature. Also it has a slight performance overhead compared to non-transactional execution. It was not discussed if the approach would also work with Software Transactional Memory and the performane

Also it requires all kernel code to be modified to work in kernel transactions, which is a great intervention in the code and will need to be reviewd and tested intensively, which has not been done in this paper. Though they state they only touched 4000 lines, they did not implement lock elision for all types of locks and it would be necessary to conduct a full cornercase-analysis to make sure no working code might break and no security loopholes will be opened up by this. Thus, I estimate the extend of changes far higher than 4000 lines to both get kernel and userspace failure coverage, which is why I am very sceptical the feature will ever be implemented in full into the linux kernel.

The CSR approach is also usable without the kernel transaction feature, empirically measured securing 70% of all kernel failures and all userspace failures already while imposing very little overhead. The modifications necessary to only tolerate userspace-failures are far smaller making them more easy to review for security or functionality implications on other code, which makes me more optimistic that a future linux update will incorporate this feature. The paper stated their solution offers the possibility to disable kernel transactions, still making an impact on the total availability.

6.4 Statistical implications of hardware failures

There was quite some work on the statistics of hardware failures. An empirical analysis of one million consumer PCs [NDO11] reveals that as soon as one physical hardware failure occurs, the machine is a lot more likely to experience another failure within a short timespan. Just for CPU-system failures, the prior probability of a failure increases by factor 100 once the machine failed once. Another paper [ESS13] speaks of factor 5-20 regarding all kind of single node failures.

This makes it questionable if the effort to save a once failed node or CPU should be taken, or if in a server context a once failing component should be abandoned for the future, as it is likely to fail again. Thus, if this method is used in a server context, it would only be helpful in the very moment the core fails, as any longterm usage of a partially failing node is not recommended. For this short-term moment the mechanism holds an availability advantage, as only one process need to be restarted at a time and the load from the failing node can be migrated to other nodes gracefully instead of abrupt.

6.5 Cascading failures

In the paper it was already mentioned that cascading failures are likely to occur. To overcome the problems of that, the algorithm was optimized to allow another failure to happen even during the migration process and cores from another die are chosen with preference to perform the migration work. However, the remaining processes in the schedule queue of the failing core are migrated to core 0. In the paper, the drawback originating from this was discussed and it was not rated problematic, as the loadbalancer will distribute the work. However the default linux loadbalancer will schedule the work fairly amongh the cores though cascading failures are likely to happen on the same chip. Maybe it would be a good idea to modify the loadbalancer to temporarily reduce the load on a chip which just had a failure, as it could decrease the amount of potential processes shutdown in case of another failure. If there is only one chip, this is of course not possible.

6.6 Component limitations and failure model

Shalev et al. described a very specific failure model, which they wanted to tolerate. However the scope of this failure model is quite small, making it possibly less useful for real world applications. At first, they limit to CPU failures. Though being the executing unit in every computer, failures can also happen at many other places. However, it would have exceeded the scope of this paper to handle DRAM, bus, storage and other failures aswell and there are technologies to partially cope with those already.

Furthermore they limit possible failures to the actual executing units only, requiring the cache to stay intact despite being on the same chip. Their reference to real-world applications was the case of a failed transistor, which makes the whole core unable to function correctly, but they did not discuss transistors which are part of the cache. Although it was difficult to find specific literature on the probability of a cache failure, cache is taking a high portion of transistors on modern chips and ignoring the chance of it failing seems like ignoring a big part of the problem. I am unsure if it would actually have been possible to implement a solution incorporating cache failures as most modern HTM implementations work in the cache, but such a solution was neither discussed nor ruled out in the paper so I assume they did not think about it.

Another issue with their failure model is the assumption that a faulty core will not get back online again. This might be the case if the core actually burned out, but it is imaginable that a temporary bus-disconnect or a soft-error caused the core to just fail temporarily. This point of critics is related to a false positive by the FDU as discussed in Section 6.2 and could aswell lead to data corruption or crashes if the core comes back online and continues executing its previous instruction pipeline. There is no work done to prevent this nor was this issue discussed in the paper.

7 Conclusion in context

The given approach requires other means of fault-tolerance to be in place already, as the shutdown of the process currently active on the core can not be avoided. It is already mentioned that this approach can only be used with other means in place, and checkpointing is given as an example. So in the given model, each process can already tolerate spontaneous shutdown. To reduce the chances of failure for each process undeniably increases it's availability, however there is no effect on the tradeoff to consistency or latency as failure chances are still far from zero. This was never an aim of this approach, and there is a lot of research available concerning the tradeoffs between these and how to manage fault-tolerance on a more abstract level. The CSR approach makes none of these approaches obsolete and does not aim to do so. It is of undeniable but not of groundbreaking impact by reducing the chance of a full node failure, not by making it impossible.

8 Conclusion

I think the paper itself is an enrichment, as this mechanism offers a benefit in availability to any computer system. However, the writers did not see the work in a real-world application, trying to reach 100% failure tolerance instead. They failed even within their given failure model though their design of that model would not incorporate all failures of hardware chips. This striving for 100% is adding very little to the aim of increasing availability and poses several new problems in terms of overhead and feasibility. I personally would have preferred if the focus was rather to incorporate Cache-failures aswell, which could have benefitted more to the total availability improvement in a real world scenario and thus legitimizing their paper title, which suggested they aimed at commodity systems. Instead they went for the impossible, trying to make their paper a groundbreaking work in the greater context of fault-tolerance, as 100% security would have opened the doors for a rethinking in the CAP tradeoff.

At the state the work is right now, I would not recommend to incorporate the kernel-transaction approach into the commodity linux kernel, however I think the solution without transactions is a great candidate for a future linux release. I would recommend to the authors to commit two different pull requests of their solution, one just including the basic mechanism and one including everything, so the first approach could be used in the broad public and the second in cases where there is a strong focus on survival of the system, e.g. in satellites or other difficult to maintain environments.

References

[CHLVP14] C.S. Cardinell, R.G. Hathorn, B. Laubli, and T.J. Van Patten. Inter-processor failure detection and recovery, September 30 2014. US Patent 8,850,262. 6.2

[ESS13] N. El-Sayed and B. Schroeder. Reading between the lines of failure logs: Understanding how hpc systems fail. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, June 2013. 6.4

[NDO11] Edmund B. Nightingale, John R. Douceur, and Vince Orgovan. Cycles, cells and platters: An empirical analysisof hardware failures on a million consumer pcs. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 343–356, New York, NY, USA, 2011. ACM. 6.4

[Ohi09] Y. Ohira. Multi-cpu failure detection/recovery system and method for the same, December 10 2009. US Patent App. 12/544,618. 6.2

[SHP$^+$16] Noam Shalev, Eran Harpaz, Hagar Porat, Idit Keidar, and Yaron Weinsberg. Csr: Core surprise removal in commodity operating systems. *SIGOPS Oper. Syst. Rev.*, 50(2):773–787, March 2016. 4